Introduction to Programming with Python

Riding the Serpent

Anshul Nigham & Rob Tirrell

November 14, 2010

About Us

Anshul Nigham

- Began programming at 12 on on 8086 PC-XT (a full-fledged computer 20x slower than an iPhone).
- Converts caffeine into code for a living at Google.

Rob Tirrell

- In the second year of a five to six year sentence in the Butte lab (http://buttelab.stanford.edu/), an entirely 'dry' lab (computers only – the only other equipment necessary is a coffee machine).
- First language was Python, spends most days writing code in R, Python, Ruby and C++.

In Case of Emergency, Panic!

- But, really, please ask us questions – raise your hand or whatever.
- As this is an introductory course, many of you will be wondering about the same things. Be a leader, just ask and Make Your Voice Heard.

Principal Principles of Programming (1)

Computer programming *or* ***coding*** *is the process of designing, writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. This source code is written in a programming language. The purpose of programming is to create a program that exhibits a certain desired behaviour. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.*

– Wikipedia

Principal Principles of Programming (2)

~~***Computer programming*** *or* ***coding*** *is the process of designing, writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. This source code is written in a programming language. The purpose of programming is to create a program that exhibits a certain desired behaviour. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.~~

The Heart of the Matter

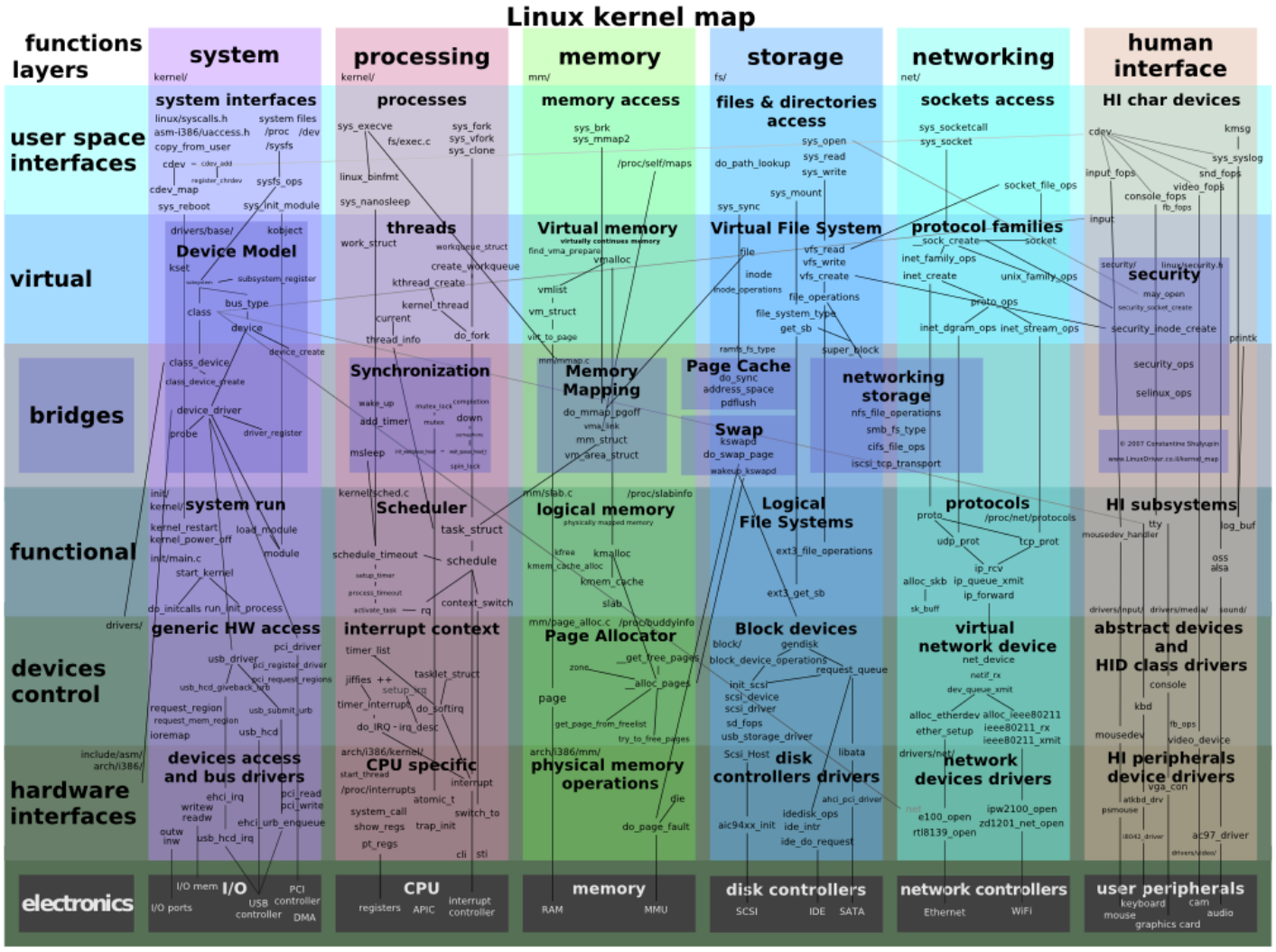
- While true, this definition is not particularly helpful, so we propose an alternative.
- At its core, programming is just writing a set of instructions which a computer runs on your behalf.
- How you conceive of and design those instructions is the fun part (and the challenge).

Principal Principles of Programming (3)

Decomposition

- **Decomposition** is the means by which a complex problem or system is broken down into parts that are easier to understand, program and maintain.
- This is one of the fundamental principles of design and programming: we decompose the logical structure into smaller, reusable units which we build one-at-a-time.
- Sometimes, how this decomposition should proceed isn't immediately clear. Then your first step should be thinking in depth about the problem you're facing, and in what ways it can be reduced into component subproblems.

Principal Principles of Programming (4)



– Wikipedia

Principal Principles of Programming (5)

Divide and Conquer

- Imagine you are a pin manufacturer in 18th century England. To achieve maximum efficiency, you will make pins step-by-step: pounding the metal into a sheet, cutting the sheet into small strips, heating the protopins, elongating the pins, punching an eyelet, polishing the pins, etc..
- This is simple enough: just a stepwise series of transformations. More often, a computer program will be more complex, and have multiple interdependent parts.

Principal Principles of Programming (6)

Motivating by Example

- Think of an application you use frequently (a desktop application, a website, a phone application, etc.).
- What are the application and user operations we would want to support, and what are their requirements?
- With your nearest neighbors, consider and discuss how you would design this application. **Go!**

What is Python?

- First released in 1991 by a Dutchman named Guido van Rossum (GvR).
- That's Self-Appointed Benevolent Dictator for Life (SABDFL) van Rossum to the rest of us.
- An interpreted, high-level language with flexible typing.
- Currently on its third major release... in other words, it's been around the block and has withstood the test of time.

A Satisfied User

Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language.

– Peter Norvig, Director of Search Quality at Google
and Computer Science Superstar

Other Satisfied Users

- **AstraZeneca** uses Python in drug discovery pipelines.
- **Phillips'** fabrication plants are managed in Python.
- **Industrial Light & Magic** (Star Wars) employs Python for process management.
- **Anshul** developed an AdWords (Google's advertising platform) optimizer in Python.
- In **Rob's** work, people use Python at every point in the research pipeline (preprocessing and sanitization, standard analyses, data aggregation and integration, and so forth).
- It may be new to you, but according to TIOBE's programming languages index, Python is the sixth-most popular in the world.
- A list of anecdotes can't quite prove a point, so we'll try to justify why you should care about and use Python.

Good for Them, What's in it for You?

Power

- Python facilitates rapid development (quickly going from idea to implementation). It comes with a huge collection of software (libraries) for many purposes, and external libraries exist for many others.
- There is a sizeable, vibrant and very helpful Python community/ecosystem, should you run into trouble or seek advice.

Clarity

- Python is remarkably clear and readable compared to many other languages.
- It actually takes some effort to write difficult-to-understand programs.

The Interpreter (1)

>>>

- Python is an interpreted language. The computer reads, understands (compiles) and executes it on the fly, instead of reading and compiling ahead of time (as with C, C++, and many others).
- Because of this, we can use Python interactively, which is extremely helpful for designing and troubleshooting code.

IDLE

- IDLE is Python's official graphical interpreter, which we'll use to walk through examples throughout the course.
- For those of you with Macs, you can open a Terminal and launch a console interpreter by typing `python`.

The Interpreter (2)

The `print` Function

- `string`s are surrounded by single (') or double (") quotes, which are equivalent (but you must use the same type for any particular `string`). e.g., if you want a `string` to have a contraction, you can use double quotes to surround it, like `"The white whale mustn't breach, Moby is waiting."`.
- `print` is a core Python function, which by default outputs text to the screen. It's somewhat special, in that parentheses are optional when using (calling) it.

A Longstanding Tradition

Three equivalent `'Hello, World!'`s:

```
1  print 'Hello, World!'
2  print "Hello, World!"
3  print("Hello, World!")
```

The Interpreter (3)

Not Your Daddy's Desktop Calculator

- Among other uses, we can employ Python as a calculator.
- At the interpreter (`>>>`), type `4 + 2`. You should see `6`. So far, so good – naturally, other operators are available (e.g. `+ - * /`).

Balling

- To delve deeper into Python, we're going to work through a baseball[1] statistics[2] example.

[1] Baseball is a popular American sport. Rob's hometown is the supposed 'birthplace of baseball'.

[2] The deliberate application of statistics to baseball is known as 'sabremetrics'. Check it out, it's really interesting.

The Interpreter (4)

Still Balling

- The number of bases a player moves as a result of his offense is calculated as:

  ```
  singles + 2 * doubles + 3 * triples + 4 * homers
  ```

- In `1920`, Babe Ruth appeared at the plate `458` times, cranking out `73` singles, `36` doubles, `9` triples and `54` home runs.

The Interpreter (5)

More than a Glorified Calculator – Saving it for Later

- It can be tedious to have to type out that expression every time. Luckily for us and our fingers, we can store the results of calculations in **variables** very simply in Python.
- For example,

  ```
  bases = 73 + 2 * 36 + 3 * 9 + 4 * 54
  ```

  If you mess up the numbers, you can always try again. The new value will replace the old one.
- Voilà! Try `print`ing the result.

The Interpreter (6)

Back to the Babe

- Another useful statistic is **slugging percentage** (SP):

```
1   (singles + 2 * doubles + 3 * triples + 4 * homers) / at_bats
```

- Since we've already calculated the numerator — the number of bases the player moves as a result of his offense — it's trivial to calculate the slugging percentage. [3]

```
1   bases = 73 + 2 * 36 + 3 * 9 + 4 * 54
2   sp = bases / 458.0
3   print sp
```

[3] The divisor – at bats – needs to be a floating-point number, so enter it as `458.0`, otherwise Python will drop the fractional component of the result. This behavior can be a pain, and has been changed in Python 3.

Brief Interlude: Setting Up KomodoEdit

New Release!
ActiveState
Komodo® Edit 6

- KomodoEdit should already be installed on the provided computers, if not, visit http://www.activestate.com/komodo-edit/downloads to grab it.

Playing Ball with Datatypes (1)

What are They?

- A **datatype** refers to a location in the computer's memory and the type of information stored there.
- Numbers can be of the `int` (integer) datatype, like `4`, or the `float` (floating point) datatype, like `4.0`).
- Text uses the `string` datatype, like `'Four score and seven years ago...'`.
- True and false values are `bool` (boolean) datatypes, in Python these are `True` and `False`. There is also a special value called `None`, that indicates 'no value'.
- We can view the type of any variable using `type`, so typing `type('bananagram')` returns `<type 'string'>`.

Playing Ball with Datatypes (2)

More Advanced Datatypes

- Obviously, more complex programs require more complex datatypes (also referred to as 'data structures').
- The most important in Python are `list`s and `dict`s (dictionaries).

```
1  teams = [
2    'Montreal Expos', 'Washington Nationals',
3    'San Francisco Giants', 'New York Yankeees'
4  ]
5  player = {
6    'name': 'Babe Ruth', 'ab': 458, 'h': 172,
7    '1b': 73, '2b': 36, '3b': 9, 'hr': 54,
8    'bb': 150, 'hbp': 3, 'sf': 0
9  }
```

Functions for Fame and Fortune (and Baseball) (1)

Python Provides Many Functions for Free

- **Functions** and **methods**[4] are procedures that work on variables to transform or otherwise alter them.
- Many of Python's collection datatypes support element access by index (indexing):

```
player['name'] # 'Babe Ruth'
```

- This is an important point: in Python, the first element of a collection is the `[0]`th" one, available at `collection[0]`.
- Similarly, for lists:

```
teams[0] # 'Montreal Expos'
```

[4] There is a difference: methods are attached to and operate on their objects, while functions stand alone. Don't worry too much about this now, we'll come back to it later (and in this course we will only write functions, anyway).

Functions for Fame and Fortune (and Baseball) (2)

A Few More Examples

- Change a `string` to all uppercase:

```
teams[0].upper() # 'MONTREAL EXPOS'
```

- Add an entry to a `list`:

```
teams.append('Krypton Krushers')
```

- Compute the `sum` of a `list` using the builtin `sum` function:

```
sum(player['h'], player['bb']) # 322
```

- `return` the `len`gth of a collection:

```
len(teams) # 4, or 5 counting the Krushers.
```

Functions for Fame and Fortune (and Baseball) (3)

Writing Your Own with `def`

- The `def` keyword is used to create new functions.
- Functions always `return` a value. Functions not explicitly `return`ing a value (have no `return` statement) implicitly `return None`.

```
1   # This function has no parameters and returns None.
2   def test_func():
3     print "I'm useless!"
```

Functions for Fame and Fortune (and Baseball) (4)

Funcball

- If we represent a player as in the `dict`ionary above, we can then write functions that calculates any player's stats.[5]

```
1  def calculate_bases(player)
2    return player['h'] + 2 * player['2b'] + \
3      3 * player['3b'] + 4 * player['hr']
4  def calculate_sp(player):
5    return calculate_bases(player) / float(player['ab'])
```

- Functions exist to coerce `string`s like `'1024'` into the corresponding floating-point or integer numbers (c.f. the `int` and `float` functions), or numbers into the corresponding `string` (c.f. the `str` function).
- We use the `float` functions here to account for Python's division behavior, as explained earlier.

[5] The `'\'` just allows us to continue an expression on the next line, so it doesn't run off of the slide :).

Loops (1)

Loopball

- Suppose we have all of the players in baseball stored in a `list` (the elements of the `list` are `dict`ionaries representing players).
- We could calculate their stats one-by-one, but that is just wrong. Instead, we have the `for`-loop construct:

```
1  for player in players:
2    sp = calculate_sp(player)
3    print player['name']
4    print 'Slugging percentage: ' + str(sp)
```

- Then the basic syntax is:

```
1  for item in collection_of_items:
2    do()
3    stuff = 2 + some + items
```

Loops (2)

Whileball

- Another type of loop is available to us: the `while` loop:

```
1  prompt = 'Enter your batting average: '
2  batting_average = 0
3  while batting_average <= 0.350:
4    # Convert the string from the user into a float.
5    batting_average = float(raw_input(prompt))
6  print 'Your average is >= 0.350.'
7  print 'But you already knew that...'
```

- Then the basic syntax is:

```
1  while some_condition_is_true:
2    perform()
3    any_number = of_operations
```

Loops (3)

ABA: Average Batting Average

- How about a more useful example:

```
1   print 'Give me some averages!'
2   print "Enter a negative number when you're done."
3
4   prompt = 'Enter a batting average: '
5   batting_average = 1
6   batting_averages = []
7
8   while batting_average > 0:
9     # Convert the string from the user into a float.
10    batting_average = float(raw_input(prompt))
11    batting_averages.append(batting_average)
12
13  print sum(batting_averages) / len(batting_averages)
```

Rolling Your Own Classes

Custom-Created Classes

- We can also write custom `class`es with the `class` keyword, with which we can define datatypes (or data structures) that store arbitrary data and have a set of methods to manipulate it.[6]

```
1  class MyTestClass:
2    def __init__(self, name):
3      self.name = name
4    def say_hello(self, other_name):
5      print self.name + ' says hi to ' + other_name
6  my_test_class_instance = MyTestClass('Rocko')
7  my_test_class_instance.say_hello('Clarissa')
8  # 'Rocko says hi to Clarissa'
```

- The ability to extend the language with our own `class`es is extremely useful, but for concerns of time we'll just dangle the idea in front of you and leave it that. Any book on Python will have a more thorough treatment of `class`es.

[6] **Functions** versus **methods**: `say_hello` is an example of a `method`, it's associated with `MyTestClass` objects.

Dealing with Uncertainty (1)

Truth Testing

- Few programs are just a straight pipeline of unbiased transformations to data – most of the time, we'll want to branch out and make decisions based on input.
- Python has `if`, `elif` and `else` keywords that provide this ability.

```
1  some_boolean = True
2  if some_boolean:
3    print 'Truly today is a special day!'
4  else:
5    print 'Back to the grind, peasant.'
```

It's a special day!

Dealing with Uncertainty (2)

Equality

- We can also use ==, <, <=, and so on.
- The instructions following an `elif` keyword are only triggered if no previous conditions are met and the `elif` condition is.

```
1  strength = 15
2  if strength >= 15:
3    print 'You pull the sword from the stone.'
4    print 'All hail King Arthur!'
5  elif strength >= 10:
6    print 'Solid effort.'
7    print 'But you were born to work in the mines.'
8  else:
9    print 'Nope! Not even close.'
```

All hail the king...

if-elif-else

We can also try to imagine this in a more intuitive fashion, as a hierarchical list of instructions. Then it's much more clear what the logical flow is.

`strength >= 15`?

Yes

You're King Arthur.

No

`strength >= 10`?

- **Yes**: to the mines.
- **No**: not even close.

Exercise 1

Odd or Even?

- The modulus operator `%` gives you the remainder of a division of two `int`egers.

```
10 % 3 # The remainder when dividing 10 by 3 is 1.
4 % 2 # The remainder when dividing 4 by 2 is 0.
```

- You can get the user to enter an input number as follows:

```
user_number = int(raw_input('Enter a number: '))
```

Write a program that asks the user to input a number, then `print`s out if the number is odd or even.

Exercise 2

Am I Overweight?

- Get the user to enter his or her height in inches and weight in pounds.
- Compute his or her **body mass index**[7] (BMI) as:

```
703 * weight / h ** 2
```

- Remember to use the `float` function to convert the user's `string`s to `float`s.

[7] BMI's utility as a measure of health is somewhat limited – it does not necessarily indicate fitness. For example, due to the mass of their muscles, professional athletes are often called 'overweight' by BMI.

Exercise 3

Odd and Even Averages

- Get the user to keep entering numbers until he or she enters a 0. Compute the average of all of the numbers the user entered.
- Hint 1: remember the average of batting averages? It'll look a lot like that.
- Hint 2: use `append` and `while`!
- Hint 3: use two builtin functions we've seen to easily compute the average, as we did with batting averages:

```
1   def average(list_of_numbers):
2     # Make sure that the length is calculated as a float to
3     # prevent Python from performing truncating integer division.
4     return sum(list_of_numbers) / float(len(list_of_numbers))
```

Case Study: The Candy Shoppe (1)

A Profitable Business Proposition

- You run a candy shoppe, selling all variety of popsticks, licorice screamers, fireballs and the like.
- Little kids are big business you, and these little sugar fiends are spending their parents money and are not very sensitive to changes in price. Being a businessperson first and foremost concerned with the bottom line, you want to charge anyone less than `14` years old a little bit extra (`25%` over regular price).
- But you're not a monster, and if the customer's blood sugar is below `2.5`, you'll give him or her a `15%` discount (with no additional penalty for being less than `14`).
- You want this to be done transparently, so you will add a function to your cash machine that jacks up the price depending on the customer's information (which is stored in the customer's loyalty card).

Case Study: The Candy Shoppe (2)

Customer Information

- Each customer's loyalty card has several attributes encoded in the magstripe.
- These attributes are loaded into the cash machine when you scan the card, and are available to the software as a `dict`ionary.

```
1  customer_info = {
2    'name': 'GI Jane',
3    'email': 'jane@army.mil',
4    'age': 38,
5    'blood_sugar': 5.0 # mmol / L
6    'likes_dark_chocolate': True,
7    'astrological_sign': 'Gemini'
8  }
```

Case Study: The Candy Shoppe (3)

Implementation

- Recall that we access dictionary values by their key, so for the above example, `customer_info['age']` will `return 38`.
- We can make comparisons with `dict`ionary values directly, without storing the value in a separate variable:

```
1  if customer_info['age'] > 65:
2    print 'Senior citizen discount!'
```

- We want to write a function, that given the purchase price and a customer's information, 'adjusts' the price appropriately. It should `return` a number and look something like:

```
1  def compute_price(base_price, customer_info):
2    pass # Fill in here!
```

Importing and Using Modules (1)

- So far, we've only played with the core of Python, so let's dive into the standard library (the suite of useful software that comes with Python).
- To load any one of these libraries, we use `import` statements:

```
1  import random
2  # Generate an integer from 1 to 6 (inclusive).
3  random.randint(1, 6)
4  # Generate an integer from 1 to *5* (exclusive).
5  random.rangrange(1, 6)
```

Importing and Using Modules (2)

Who's Feeling Lucky?!

- Let's use `random` to pick the winner of a lottery. First, define the players:

```
1   lottery_players = [
2     'Rufus', 'Sun', 'Tania',
3     'Boris', 'Rocky', 'Daisy'
4   ]
```

- Then, picking a winner with `random` is stupid simple:

```
print random.choice(lottery_players)
```

Importing and Using Modules (3)

Another Way

- We can also pick a random player by index in the `list`:

  ```
  random.randrange(0, len(lottery_players))
  ```

  There are a few things to note here.
- The `len`gth of the `list` is one greater than the last position you can index to. The last is then:

  ```
  lottery_players[len(lottery_players) - 1]
  ```
- Since `randrange` generates a number exclusive of its second argument, we can just use the `len`gth as the second argument.[8]

[8] Actually, we can also omit `0`, `randrange` or `randint` with one argument are assumed to start at `0`.

Importing and Using Modules (3)

Top Three

For example, we could use this to pick a `list` of three winners:

```
1   winners = []
2   while len(winners) < 3:
3     windex = random.randrange(0, len(lottery_players)
4     winners.append(lottery_players[windex])
5     lottery_players.pop(windex)
```

Three Twitter Trials

- We will use a simple library[9] to read any public Twitter feed. Download the `smalltweetlib.py` file from https://github.com/rtirrell/PythonIntroduction/raw/master/Source/twitter/smalltweetlib.py.
- Create a `.py` file in the directory you save `smalltweetlib.py` to, and import `get_tweets` at the top:

  ```
  from smalltweetlib import get_tweets
  ```

  This provides a function `get_tweets` which `return`s a list of `dict`ionary objects with text and date attributes.
- To get started, try this simple command:

  ```
  print get_tweets('cnn')
  ```

  You can also run it directly (i.e., `python smalltweetlib.py`). It will output tweets from CNN and the NY Times, ordered by date.

[9] The internals of the file are omitted from discussion today. They're slightly more advanced than we can handle. Besides, most of the time, you wouldn't want to write that sort of code yourself. Use a library!

Your First Trial

- `print` a tweet from any user (for example, `'cnn'`, or any Twitter user you choose) in a nice format on screen.
- Examine the output from `get_tweets` – it's a `list`, so you'll want to give a single tweet as the argument to your formatting and `print`ing function.

```
1  tweets = get_tweets('cnn')
2  tweets[0]
3  # An example tweet dictionary:
4  {
5    'date': u'20101113000752',
6    'text': u'robtirrell: @zimmeee sup dawg how was lunch?'
7  }
8  print_formatted_tweet(tweets[0])
```

Your Second Trial

- Display the tweets from a user (for example `'CNN'` or `'NYTimes'`) in that nice format on screen.
- Hint: use function you `def`ined above and a loop!

Your Second Trial

- Display the tweets from a user (for example `'CNN'` or `'NYTimes'`) in that nice format on screen.
- Hint: use function you `def`ined above and a loop!
- Spoiled:

```
1  tweets = get_tweets('THE_REAL_SHAQ')
2  for tweet in tweets:
3    print_formatted_tweet(tweet)
```

Your Third and Final Trial

- Display the tweets from a `list` of users in a nice format.

```
# A list of tweeps: twitter peeps.
tweeps = [
  'cnn',
  'nytimes',
  '50cent', # Vulgarity warning.
]
```

- Hint: use a loop inside a loop. It's totally legit.

```
# A list of lists of major airports in NY, CA and IL.
state_airports = [
  ['LAG', 'JFK', 'ALB'],
  ['LAX', 'SFO'], ['MDW', 'ORD']
]
for state in state_airports:
  for airport in state:
    print airport
```

As Simple As...

```
1   for user in tweeps:
2     tweets = get_tweets(tweep)
3     for tweet in tweets:
4       print_formatted_tweet(tweet)
```

The Final Project: Mangling Mark Twain

- We will code a 'random writer' that reads a document and generates a somewhat sensible-sounding babble based on it.
- This will be the whirlwind outro: we'll introduce user input and file input/output. The result is pretty cool:

> *By jingo! that reminds me of a droll dog of a thieving line– the high standard held up to the mode in which her mouth and brow there was something alien and ill-understood the impetuosity. His reason for hastening it–if he scrupulous explorer to be saluted with those cheerful view of all the plans, and took lasting impression of Celia, Tantripp, stooping and getting a bit of a note saying you don't see anything of that sort of challenge me.*
>
> – Generated from Middlemarch by Mary Ann Evans

- Try as you might, you can't make your mind understand it. It's gramatically close to English, but semantically meaningless gobbledigook.[10]

[10] Kind of like Perl :).

random_writer.py

- The file is available on the GitHub website.[11]
- It is meant to be run from the command line. Type `python random_writer.py`, and it'll ask you to input the name of the file containing the document to mangle. `Middlemarch.txt` is available in the same directory as the code. You can find other books and corpuses at Project Gutenberg (http://www.gutenberg.org/wiki/Main_Page).
- The file is reasonably well-commented, and introduces a few advanced features: `list` comprehensions and `lambda`s.

[11] Reminder:
https://github.com/rtirrell/PythonIntroduction/tree/master/Source/random_writer/

So We've Sold You on Python, What's Next? (1)

There are **tons** of freely-available resources on the internet to continuing experimenting and learning. Two books that provide particularly good starting points:

- Anshul endorses A Byte of Python (http://swaroopch.com/notes/Python).
- Rob endorses Dive Into Python 3, a slightly more advanced introduction (http://diveintopython3.org). As the name indicates, it covers Python 3 (we are using 2.7, some things differ but most of what we've gone over is the same). There's also a Dive into Python covering 2.4, but that's getting a bit dated.

So We've Sold You on Python, What's Next? (2)

- We haven't even nearly covered the standard library (the stuff that comes with every version of Python). Honestly, I haven't used or even heard of most of it.
- Python has builtin modules for:
  - Scraping from the internet (`HTMLParser`).
  - (Extremely advanced) `string` searching and manipulation (the strange world of regular expressions, in the `re` module).
  - Creating, deleting and managing files and directories (`os` and `shutil`).
  - Creating and extracting compressed files (`gzip`, `bz2`, `zipfile`, `tarfile`).
  - Interacting with FTP servers (`ftplib`).
  - And so on and on $\rightarrow \infty$.

So We've Sold You on Python, What's Next? (3)

Stepping away from builtin functionality, there are many other mature libraries in the wild:

- If you want to write websites, check out Google's App Engine (http://code.google.com/appengine/). It provides free hosting services and holds your hand along the way (that's a good thing!). For more advanced websites, look into the Django Project (http://www.djangoproject.com/) (App Engine is actually based on Django).
- There are libraries for image manipulation (the Python Imaging Library, http://www.pythonware.com/products/pil/), numerical and scientific calculation (NumPy/SciPy, http://numpy.scipy.org/), bioinformatics (BioPython, http://biopython.org/wiki/Biopython), plus a whole slew of others for most every purpose imaginable.

... And a Universe of Other Languages

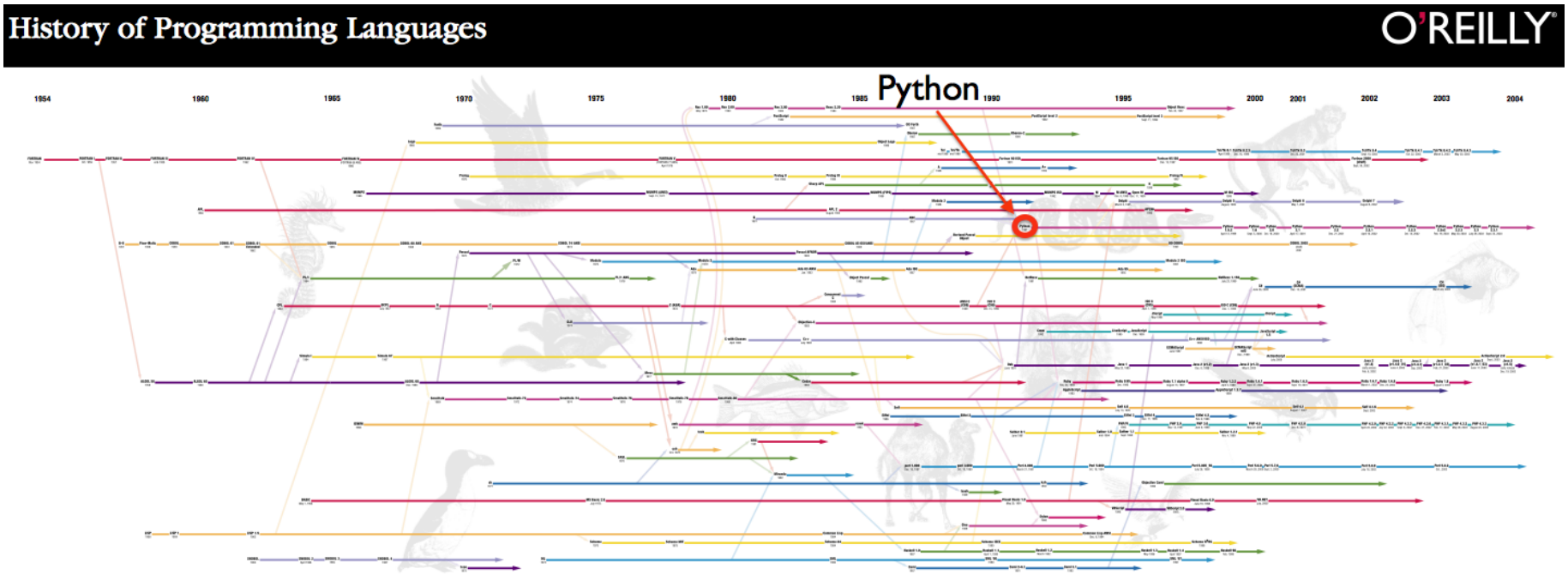


Image credit: O'Reilly (http://oreilly.com/news/graphics/prog_lang_poster.pdf)

Thanks!

- We hope this little taste of the joys of programming whetted your appetite – if so, this is the best place in the world to be.
- If you have any questions about Python, programming, or computer science in general, feel free to email either of us.
- Anshul: nigham@gmail.com, Rob: rpt@stanford.edu.
- `print 'Farewell, and godspeed!'`